On building a more efficient grammar by exploiting types

DAN FLICKINGER
Center for the Study of Language and Information, Stanford University, Stanford, CA 94305, USA
e-mail: dan@coli.stanford.edu
(Received 7 November 1999; revised 2 March 2000)

Abstract

Modern grammar development platforms often support multiple devices for representing properties of a natural language, giving the grammar writer some freedom in implementing analyses of linguistic phenomena. These design alternatives can have dramatic consequences for efficiency both in processing and in grammar building. In this paper I report on three experiments in making systematic modifications to a broad-coverage grammar of English in order to gain efficiency without loss of linguistic elegance. While the experiments are to some degree both platform-dependent and theory-bound, the kinds of modifications reported should be applicable to any unification-based grammar which makes use of types. The results make a strong case for a more visible role for the linguist in the collaborative effort to achieve greater processing efficiency.

1 Introduction

The modern grammar engineer working within the framework of a well-developed linguistic theory must make many design decisions when building a broad-coverage implementation, balancing clarity and efficiency. Any given development platform provides a range of tools for defining and processing a grammar, and linguistic theories also employ multiple formal devices which can overlap in functionality. Matching the requirements of the theory to the capabilities of the platform is in part an ongoing empirical exercise, since performance properties are likely to change as the grammar grows.

Most reported work on efficiency in natural language processing addresses better algorithms for parsing or generation, more suitable formal devices for describing linguistic phenomena, or more effective grammar development platforms. It is tacitly assumed in much of this work either that the grammars used for evaluation must be preserved unchanged, or that the grammar writer will cooperatively and quietly make the best use of the tools available. Rarely are the evaluations of alternative grammatical representations reported for a given grammar on a given platform, though many such empirical decisions must be part of the development of any broad-coverage grammar. As one notable exception, see Verlinden (1999) for
a detailed discussion on constructing an efficient grammar in ALE. Emelé (1997) includes in his development of the TFS formalism a useful discussion of efficiency issues in the choice of macros\(^1\) versus types to represent regularities such as the feature principles of HPSG. Butt, King, Nino, & Segond (1999) offer a detailed discussion of engineering issues within the ParGram project, but do not explore efficiency-driven alternatives in the design of the grammars. Brief mentions of efficiency tradeoffs in semantic construction can be found in Alshawi (1992), but no detailed discussion of alternatives explored and rejected during development of the CLE grammar.

In this paper I report on three experiments in varying the choice of grammatical representation within a broad-coverage grammar, in order to achieve improved efficiency without changing the engines used for running the grammar. Naturally, such experiments often reveal ways in which a processing engine can be improved, especially for a particular grammar, but the goal here is to disentangle the two parts of that synergistic development cycle, to see what decisions the grammar writer can make to maximize efficiency given some set of descriptive devices, without compromising the elegance or maintainability of the linguistic representation.

I begin with descriptions of the grammar and the two development systems used in the experiments, then describe each experiment in turn, to provide an empirical basis for the position that the grammar writer should play a central and visible role in the effort to produce a precise broad-coverage grammar that can be processed efficiently.

2 Grammar and systems

2.1 The LinGO English Grammar

The English Resource Grammar, under development in the Linguistic Grammars Online (LinGO) project at the Center for the Study of Language and Information, is an implementation of a large grammar fragment written within the linguistic framework of Head-driven Phrase Structure Grammar (HPSG; Pollard & Sag, 1994).\(^2\) While still closely aligned with the variant presented in Chapter 9 of that work, the CSLI implementation contributed to and elaborates on the enriched inventory of syntactic rules motivated in Sag (1997). The grammar also varies from the standard Pollard and Sag framework in its use of Minimal Recursion Semantics (MRS; Copestake, Flickinger, Malouf, Riehemann, & Sag, 1995) for specifying semantic attributes throughout, and it lacks any implementation of binding theory. In most other respects, the implementation attempts to remain faithful to the theory of grammar that has been built up based on Pollard & Sag (1994).\(^3\)

---

\(^1\) Here, an abbreviation for a definition of a feature structure, provided for the convenience of the grammar writer (page 126).

\(^2\) The grammar in its current form is the product of significant collaboration with a number of my colleagues at CSLI, particularly Emily Bender, Ann Copestake, Rob Malouf, Ivan Sag, and Jeff Smith.

\(^3\) One further departure from the standard formalization is a restriction in the grammar to only binary and unary branching phrase structure rules, in contrast to Pollard and
On building a more efficient grammar by exploiting types

The LinGO English grammar (January 2000 version) consists of a rich hierarchy of 1,663 abstract lexical types, a smaller hierarchy of 180 rule types, 58 specific phrasal and lexical rules, and 6,766 lexical entries. The grammar uses 79 features defined on the lexical and phrasal types, plus an additional 39 features for the various arguments of the 4,562 lexical semantic predicates. The text files that define the grammar, excluding the lexical entries, comprise 12,600 lines written in the Type Description Language TDL (Krieger & Schäfer, 1994).

As one measure of breadth of coverage, this grammar provides correct syntactic and semantic analyses for 83 per cent of the 8,520 well-formed utterances found in the transcriptions of 175 human-human dialogues recorded as part of the Verbmobil spoken language translation project (Bub, Wahlster, & Weibel, 1997). Sentence length in this sample averages 8.5 words, with the conversations on the topics of scheduling meetings and making hotel reservations. Lexical ambiguity in this sample averages 2.2 lexical entries per word excluding the application of lexical rules, with wide variation in lexical ambiguity between closed-class and open-class lexical entries.

2.2 The PAGE platform

The first of two grammar development environments that we used to build the LinGO English grammar was the Platform for Advanced Grammar Engineering (PAGE), developed in Common Lisp at the DFKI (German AI Research Center) in Saarbrücken, Germany. PAGE, formerly known as DISCO (Uszkoreit et al., 1994), includes a language and interpreter for high-level definition of typed feature structures in a multiple inheritance hierarchy (TDL), a parameterizable chart parser, and a set of analysis and debugging tools including a feature structure viewer (FeGramEd) and a parse chart viewer. The relevant capabilities that we experimented with are introduced as needed in the discussion below. PAGE has served both for our grammar development and for efficient parsing of spoken input using the grammar within the Verbmobil translation system (Kiefer, Krieger, Carroll, & Malouf, 1999).

2.3 The LKB system

In parallel with our work on coverage for the Verbmobil project, we have been developing a research prototype at CSLI of an efficient generator for use by disabled people as part of a speech prosthesis (Carroll, Copestake, Flickinger, & Poznanski, 1999). The generator is part of the Linguistic Knowledge Building (LKB; Copestake, 1999) system, which also includes a chart parser, a unifier, and the necessary analysis tools for viewing feature structures, parse charts, and type hierarchies. For the Sag’s flat structures for e.g. double-object verbs or inverted sentences. Our variant produces more nodes in the analysis of some sentences, but reduces the number of rules in the grammar without requiring Kleene operators, and enables a simpler account of unbounded dependencies.

The present-day LKB is a descendant of the Lexical Knowledge Base system originally developed as part of the ACQUILEX project (Copestake, 1992).
purposes of the experiments reported here, we took advantage of the LKB’s parser and its ability to read in the grammar definitions in the same TDL syntax used for PAGE. The LKB differs from PAGE in requiring that each feature be introduced on exactly one maximal type, thus enabling type inference (see the Appendix by Copestake, this volume).

The parsing strategies employed by PAGE and the LKB are described and contrasted in Oepen & Carroll (this volume), but the details of those differences are not relevant to our discussion, since the goal of the experiments reported here is to identify and evaluate choices that the grammar writer can make with a given processing engine. By using two similar but quite distinct bottom-up parsers which employ only unification to combine lexical and phrasal signs (typed feature structures), we sought to exploit the best features of these existing engines, while avoiding the effects of over-specializing grammar to parser and vice versa.⁵

3 Experiment 1: Eliminating disjunctive feature specifications

An early design decision made in the development of the LinGO English grammar was to make use of the considerable expressive power of disjunctions and negations of feature values that was provided in TDL. But performance measurements showed that a significant improvement in speed at run-time on the PAGE platform could be achieved if the unifier did not permit disjunctive feature values. The elimination of disjunctive values from a broad-coverage grammar is a radical solution to the problem of efficient processing of disjunctive values in unification. An active line of research since the mid-1980s (see, among others, Kasper, 1987; Eiselle & Doerre, 1988; Nakano, 1991; Emelé, 1991; Maxwell and Kaplan, 1995). In all of this work it is assumed, and in some cases argued (Karttunen, 1984. for English and German. Carter, 1990. for Japanese) that disjunction is an essential device for expressing linguistic generalizations; and indeed, disjunction is frequently employed in published HPSG analyses of phenomena for English as well. During much of the 1980s, work on unification did not make use of types, but once types are introduced as a tool for grammatical description, they present the option of reducing or even eliminating the use of disjunctions. Our exploration of this option was encouraged by the high processing cost for using disjunctions within the PAGE system, leading us to re-examine the actual uses of disjunctive values in our grammar for English.

We found that in almost every case, we could express the relevant linguistic generalizations in the grammar at least as well by making better use of the type hierarchy. Indeed, in several instances the choice between disjunction and type underspecification seemed entirely arbitrary from a descriptive viewpoint. So we revised the full grammar to eliminate all use of disjunction in feature values, in most cases by enriching the type hierarchy with new underspecified types whose subtypes were the members of the original disjunction. In some cases, a more interesting

⁵ This is a real and well-recognized danger; see for example Carroll (1994), who notes the asymmetry in performance for a grammar run with the Alvey Tools parser and with the CLARE parser.
reanalysis of the type hierarchy emerged. For example, in the earlier grammar the constraint on person-number agreement for non-third-singular present-tense verbs was expressed as the following disjunction for the value of the AGR(eement) feature:

\[
(1) \begin{array}{c}
\begin{array}{c}
\text{non3sg-verb} \\
\text{AGR}
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
\{\text{NUMBER sing}\} \\
\text{PERSON}\{\begin{array}{c}
\text{first} \\
\text{second}
\end{array}\}
\end{array}
\end{array}
\begin{array}{c}
\{\text{NUMBER plur}\}
\end{array}
\]
\]

However, in the entry for the idiosyncratic past-tense copula was the same grammar already employed a more abstract type for the AGR value, suggesting the more general solution we would adopt for these disjunctive values:

\[
(2) \begin{array}{c}
\begin{array}{c}
\text{sg-past-cop} \\
\text{AGR}
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
\{\text{NUMBER sing}\} \\
\text{PERSON first-or-third}
\end{array}
\end{array}
\end{array}
\]

The values of the agreement features PERSON and NUMBER were types arranged in the simple hierarchies shown in the left-hand side of Figure 1. To eliminate the disjunction used for non-third-singular-agreement verbs, we developed instead the single hierarchy of types in the right-hand side of Figure 1, combining person and number to create types which more directly reflect the distinctions drawn in agreement for English. Given this, the AGR value for our non-third-singular verbs is simply:

\[
(3) \begin{array}{c}
\begin{array}{c}
\text{non3sg-verb} \\
\text{AGR}
\end{array}
\end{array}
\begin{array}{c}
\{\text{PERNUM non3sg}\}
\end{array}
\]
\]
The type non3sg will unify with any of its subtypes, including the AGR values 1sg, 2sg, 1pl, 2pl, and 3pl. Note that this hierarchy excludes some of the possible values in the cross-product of the values for number and person; for example, the grammar correctly reflects the fact that there is no motivation for a first person type independent of number in English.

Analogous restructuring of the type hierarchy served to eliminate most of the remaining several dozen uses of disjunctive values in the LinGO English grammar, yet allowed us to retain direct representation in the grammar of the relevant linguistic generalizations. In addition to the compactness of representation that types afford, the transparency of description in the run-time grammar further distinguishes this type-based approach from compilation-based alternatives which eliminate disjunctions at run-time.

In two cases, both syntactic rules, we had encoded a more complex distributed disjunction where the values of two features co-varied, and in each case we split the rule into two more specific subtypes, in order to fully eliminate the use of disjunctive values within the structures. For example, the rule for filler-head phrases with a WH-filler ensured that the value of the boolean feature INV corresponded correctly with the value of the boolean feature root, so that matrix WH-questions were inverted, while embedded WH-questions were not. The slight increase in the number of rule instances resulting from these two splits incurred no measurable additional cost in processing.\(^6\)

While I am not prepared to argue that disjunctive values can and should be eliminated from all linguistic description, this experiment demonstrates that a significant fragment of English can be described without that formal device, given the expressive power of the multiple-inheritance type hierarchy already assumed in HPSG. The revised grammar retained at least an equivalent clarity in linguistic description, and produced the same syntactic/semantic analyses as the original grammar when evaluated against both our standard 1350-item test suite of typical English phenomena, as well as the 4511-item TSNLP English test suite (Oepen, Netter, & Klein, 1997).

Given this satisfactory result linguistically, it was gratifying to see a significant gain in efficiency for the revised grammar running on PAGE. Table 1 shows the difference in performance for the English grammar with and without disjunctive value specifications, measured against both the TSNLP English test suite and the 96-item ‘agad’ test set, a small sample from the Verbmobil corpus (see the introduction to this volume for details on the test data), running on the same machine.\(^7\)

The two rows reflect the two runs of the test suite on the same hardware with the

\(^6\) Indeed, our experience with a modestly increasing inventory of rule instances to the current several dozen is consistent with the results obtained by Nagata (1992), who found improved efficiency after hand-converting a ‘course-grained’ unification grammar of 22 rules to an equivalent ‘medium-grained’ grammar of 164 rules as part of the ATR SL-TRANS project.

\(^7\) The data for this and all other measurements reported here was collected and analyzed using the [linr tshb] performance profiling tool that provides uniform benchmarking facilities for PAGE, the LKB, and a number of additional platforms (Oepen & Flickinger, 1998; Oepen & Callmeier, 2000).
two versions of the grammar, one from October 1996 and one from October 1997. The columns record, from left to right, the number of test items (sentences) in each run, the average number of executed tasks for each sentence (that is, the number of attempts to unify two signs), the average number of seconds required to process each item exhaustively (all parses), and the average number of megabytes of heap consumed in processing each item. The later grammar still preserves essentially the same analyses for the examples covered by the earlier grammar, but has somewhat broader coverage and a larger lexicon; hence the larger number of executed tasks (attempted unifications). For the TSNLP test suite, we see an increase of 1.5 times the number of unifications, but the overall time dropped by a factor of 3.5, indicating that the cost per unification actually dropped by a factor of 5.25, given the close correlation between number of executed tasks and overall resource consumption. The gain in efficiency without disjunction is even clearer with the Verbmodul (‘aged’) data, where we see the wider coverage grammar execute 2.3 times more tasks than the earlier grammar, but reduce the time required per item by 3.9, showing that the cost of unification dropped by a factor of 9.0. Some significant part of this speed-up is due to the efficient non-disjunctive unifier developed by Rob Malouf, one of the improvements reported in Kiefer et al. (1999) and further described in Malouf, Carroll, and Copestake (this volume); this unifier has been incorporated into both PAGE and the LKB. But of course this streamlined unifier could only be used once the grammar was free of disjunctions. Note also that space requirements (memory consumption) also dropped significantly when using the non-disjunctive grammar and unifier; since this is a Lisp-based system, where garbage collection can be costly in time, the reduction in memory usage contributes quite directly to real time reduction, especially when profiling over large data sets.

This efficiency gain required a detailed analysis and manual revision of the grammar, though one that was driven as much by linguistic motivations as it was by concerns for more speed. If the only motivation had been speed, and especially if the resulting grammar had lost descriptive clarity in the process, one might well ar-

<table>
<thead>
<tr>
<th>Platform</th>
<th>Test suite</th>
<th>Grammar version</th>
<th>items</th>
<th>etasks</th>
<th>time (s)</th>
<th>space (kb)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAGE</td>
<td>TSNLP</td>
<td>with disjunction</td>
<td>4,511</td>
<td>656</td>
<td>3.22</td>
<td>18,016</td>
</tr>
<tr>
<td>PAGE</td>
<td>‘aged’</td>
<td>with disjunction</td>
<td>96</td>
<td>1,684</td>
<td>32.8</td>
<td>68,629</td>
</tr>
<tr>
<td>PAGE</td>
<td>TSNLP</td>
<td>no disjunction</td>
<td>4,511</td>
<td>1,006</td>
<td>0.92</td>
<td>5,747</td>
</tr>
<tr>
<td>PAGE</td>
<td>‘aged’</td>
<td>no disjunction</td>
<td>96</td>
<td>3,885</td>
<td>8.4</td>
<td>24,373</td>
</tr>
</tbody>
</table>

Table 1. Performance on two test suites using October 1996 LinGO grammar with disjunctions, contrasted with October 1997 version where disjunctions have been eliminated. The first two lines of data are for the TSNLP English test suite, and the second pair of lines show data for 96 somewhat longer examples (8.4 word average) from the ‘aged’ test set.
gue that compilation could have been a better alternative. Several efforts to relieve the linguist from responsibility for efficiency through the use of macros or compilation of HPSG grammars have been pursued; Götz & Meurers (1995) report on compiling into DCGs, Kasper, Kiefer, Netter, & Vijay-Shanker (1995) on compiling into TAGs, and Pulman (1996) on the creative use of macros. But given our central concern for efficiency in the development and testing cycle itself, it is undesirable to introduce time-consuming compilation, or strategies that introduce opacity between the representations the grammar writer thinks in and those used by the processor. This need for transparency and rapid turn-around when extending or debugging a large grammar has been noted before (e.g. by Boguraev, Carroll, Briscoe, & Grover, 1998, reporting on experience with modest compilation for the Alvey Tools grammar), but is often omitted from investigations of efficiency reported by NL system builders.

4 Experiment 2: Reducing the number of lexical entries

Given the decision to avoid the use of disjunctions within feature structures, a naïve treatment of lexical variation could result in a large number of closely related but distinct lexical entries, to cope with phenomena such as optional complements, alternative subcategorization frames, and ambiguity in lexical semantics. The widespread use of lexical rules in HPSG analyses to account for syntactic phenomena such as extraction or inversion further magnifies the potential cost in efficiency at run-time (Noord & Bouna, 1994; Meurers & Minnen, 1995). More lexical entries can create more search space that a parser must explore to produce the analyses licensed by the grammar for a given input, and if those alternatives grounded in lexical ambiguity are not resolved quickly, a large number of unproductive edges will be constructed, at some cost in both time and space.

One potential source of lexical ambiguity arises from the need to express alternations in subcategorization due to optionality of arguments. For example, the ditransitive verb ask takes three arguments, two of which are optional:

(4) a Kim asked Sandy a question.
    b Kim asked Sandy.
    c Kim asked a question.
    d Kim asked.

Having eliminated the possibility of disjunctive values, we have two basic choices for representing this kind of alternation: (1) move the disjunction to the lexicon, providing four separate lexical entries for this sense of ask; or (2) encode optionality directly as a property of subcategorized-for elements, and modify the grammar to accommodate this property. Pursuing the latter solution, I introduced a boolean feature opt as a property of each element in the valence lists for complements, specifiers and subjects for each lexical entry. Our verb ask, then, has the following single entry:
I experimented with two alternatives in the way the grammar rules dealt with these lexical specifications of optionality. The first and more straightforward approach used non-branching syntactic rules to discharge missing optional elements, with one such rule each to discharge an optional complement, an optional specifier, or an optional subject. These additional rules allowed the rest of the grammar to remain essentially unchanged in how it characterized partially or fully saturated phrases: an NP, for example, has an empty specifier list and an empty complements list, while a VP has an empty complements list, but a non-empty subject list. On this approach, the resulting phrase structure trees have rather more nodes than in a typical HPSG analysis, but the desired semantics can be preserved.

The second strategy employs a new type of list as a possible value for these valence features, a type which imposes a constraint on each of its elements, namely that each element unify with [OPT +]. The subtypes of the type list are arranged as follows:

\[(6) \]

\[
\begin{align*}
\text{list} & \quad \text{neist} & \quad \text{optlist} & \quad \text{elist} \\
\text{opt-neist} & \quad \text{FIRST} \quad [\text{OPT +}] & \quad \text{REST} \quad \text{optlist} \\
\end{align*}
\]

With this definition of optlist, valence features in lexical and phrasal types that were formerly assigned the empty list as a value are instead given the value optlist, which will unify either with the empty list as before, or with a non-empty list each of whose elements unifies with the constraint [OPT +]. On this approach, it is not necessary to discharge optional elements to produce a saturated sign, so fewer nodes have to be constructed by the parser from lexical heads with missing optional arguments. Note that the correct behavior of optlist as defined here is dependent on strict typing, which is supported in the LKB system but not in PAGE, so this second experiment was performed using the LKB.\(^8\)

Table 2 shows the difference in time and space costs for these two versions of

---

\(^8\) On this approach, it is also necessary to add one non-branching rule which discharges an optional complement when there is at least one obligatory complement to its right. This extra rule diminishes the elegance of the approach somewhat, but does not materially detract from performance.
the LinGO English grammar, profiled using the ‘csli’ test suite which is rich in examples testing properties of subcategorization.

The version of the grammar employing the new subtype for valence lists uses roughly thirty per cent less time and twenty per cent less heap space than the version that instead uses non-branching rules to discharge optional arguments.\(^9\) Here again, the introduction of additional abstract types provides us with a grammar that can be processed more efficiently without loss of generality. An obvious further experiment would contrast the grammar version using \textit{optlist} with one that instead employed multiple lexical entries to encode optional arguments; but it should be clear that the performance of that version would be even worse than the version using non-branching rules, since those additional entries would also interact with lexical rules to create a larger initial search space for the parser.

5 Experiment 3: Reducing size of feature structures

It is no surprise to anyone working with HPSG grammars that the feature structures for lexical and phrasal signs are large, employing dozens or even hundreds of attribute-value pairs to characterize the properties of a single sign. Within a typed feature structure framework that supports total well-typedness (see Appendix), a significant number of the features found in a sign’s description are present because they are defined for the relevant types that together comprise the sign, but they retain the most general type that was defined as their value. These most general values do not provide any additional constraint benefits during processing, yet they impose time and space costs during unification due to the extra burden of copying them upon successful unification.

One strategy for reducing this cost is to apply a filtering process on lexical and rule signs after the grammar is loaded, removing these non-informative attribute-value pairs to reduce the size of the feature structures used during processing. Götz

\(^9\) The resulting analyses for the test suite items are almost but not quite the same for the two versions of the grammar; they make slightly different predictions for sentences with constituent coordination. Even so, the difference in the number of edges constructed for these coordinate structures in the two versions is consistent with the contrasts found for the rest of this test suite.
(1993) proposes such a process, which he terms unfilling, and Gerdemann & King (1994) make use of this device to reduce the number of named disjunctions in typed feature structures. Note that to ensure correctness of the results of processing on ‘unfilled’ structures, there must also be a procedure for full reconstruction of the feature structures.

The alternative strategy I explored in this third experiment using types for greater efficiency was to reduce the need for an unfilling operation by again introducing additional abstract types. In the LinGO English grammar there are a few types like head or synsem which appear with great frequency in every sign, and they are defined with several appropriate features, so each time the attribute head is introduced, its most general value head is also introduced, and thus all of the appropriate features for head. To avoid this proliferation of most general feature value pairs, I introduced an abstract type head\textsubscript{\text{\_min}} whose only subtype is the familiar head, but head\textsubscript{\text{\_min}} itself introduces no features. Then the type cat which introduces the feature HEAD defines that feature’s most general value to be head\textsubscript{\text{\_min}} instead of head. The effect of this change is that the various appropriate features for head only get introduced when a unification occurs which either specifies a more specific subtype of head\textsubscript{\text{\_min}} or mentions one of the features of head.

To obtain the performance data shown in Table 3, I inserted into the type hierarchy similar abstract types for ten of the types most frequently mentioned within feature structures, and made the corresponding handful of adjustments in the rest of the grammar. As the data shows, it took about twenty eight per cent less time and heap space to produce essentially the same syntactic and semantic analyses using the version of the grammar with these additional minimal types.\textsuperscript{10}

While in this third experiment it might appear that the automated preprocessing strategy to ‘unfill’ feature structures is more general and more effective (Callmeier,
this volume, presents supporting evidence). I reaped an additional benefit from the manual revising of the grammar. It became clear that the idea of adding minimal types could be used to good advantage in more interesting cases such as the reification of types like “np” or “vp” which had previously been no more than macros used to define complex feature structures for subcategorized-for elements. By introducing minimal types like np with more specific subtypes that introduced constraints on the relevant features, the size of feature structures for lexical items appears to shrink rather dramatically, and the use of macros, another redundant formal device in a framework with types, can be entirely eliminated from the grammar. This more ambitious experiment with minimal types is not yet complete, but initial results are promising.

6 Conclusions
The three experiments extending the use of types in the LinGO English grammar provide clear evidence that the grammar engineer, given detailed profiling data about the performance of a grammar against a relevant test suite on a given platform, can make linguistically defensible revisions to the grammar which result in improved performance. It is clear that grammar engineers regularly carry out this kind of customization, but there are few detailed discussions in the literature of such adaptations and their consequences for the linguistic theory being implemented. Our experience with improvements in the LinGO grammar should be applicable to other grammar developers who employ typed feature structures, and underscores the importance of using tools for systematic measurement and recording of performance and competence data in the course of grammar development.

Acknowledgments
This research was funded both by the German Federal Ministry of Education, Science, Research and Technology (BMBF) in the framework of the Verbmobil Project under Grant FKZ.01IV7024, and by the National Science Foundation under grant number IRI-9612682.

References
On building a more efficient grammar by exploiting types


